Docket No. 42390.P11196
Express Mail No. EL651820743US

UNITED STATES PATENT APPLICATION

FOR

BRANCH-FREE SOFTWARE METHODOLOGY FOR TRANSCENDENTAL FUNCTIONS

Inventors:

Ping Tak Peter Tang

Prepared by:
BLAKELY SOKOLOFF TAYLOR & ZAFMAN LLP
12400 Wilshire Boulevard, Seventh Floor
Los Angeles, California 90025
(310) 207-3800

BRANCH-FREE SOFTWARE METHODOLOGY FOR TRANSCENDENTAL FUNCTIONS

Background

[0001] This invention is related to software methodologies for computing transcendental functions.

[0002] The fast and accurate evaluation of transcendental functions such as exponentials, logarithms, and trigonometric functions and their inverses, is highly desirable in many fields of scientific and engineering computing. Software implementations of these are typically written in assembly language code and use look-up tables to approximate one or more intermediate values in the computation, for faster evaluation of the function.

[0003] A typical software implementation of the general base logarithm function $\log_b(X)$ starts with representing X, a positive real number, in the floating point form $Y\,G^k$ where Y is a positive real number greater than or equal to 1 and less than G, G is a positive integer, and k (the exponent) is an integer. The limits on Y, G, and k depend on the hardware capabilities of the data processor that is executing the software. The software also includes a predefined look-up table which gives values of $\log_b(1/B_j)$ that have been previously computed for a number of breakpoints $B_0 > B_1 > \ldots > B_N$ (B_j). Depending on the input X, a breakpoint B_j is selected so that $|Y*B_j-1|$ is less than a predetermined value, delta. In general, B_j is selected to approximate $1/Y$ to a short precision. The function $\log_b(X)$ is then computed via the relationship:

$$\log_b(X) \approx k \log_b(2) + \log_b(1/Bj) + \log_b(1+(Y\,B_j-1))$$

[0004] The third term can be computed using conventional polynomial approximations. The logarithm function is implemented in this manner to improve the accuracy of the result as well as its speed of computation. This methodology is depicted by the operations 104, 108, 112, and 116 in the right-hand column of **Fig. 1** which shows a flow diagram of a conventional methodology for computing the logarithm.

[0005] Due to the finite numerical precision that is available for representing numbers in a machine, arithmetic operations performed by the machine can result in roundoff error, caused by either truncation of or rounding up (or down) a result of the arithmetic operation. Under certain situations, such as when the argument lies very close to a root of the function, alternative numerical techniques are used to limit the severity of the roundoff error. Thus, rather than follow the general relationship, in operations 104-116 described in the previous paragraph, a completely different relationship is used to compute $\log_b(X)$ when X is very close to 1. This other relationship is depicted by the operations 120, 124, and 128 in the left-hand column of **Fig. 1**.

[0006] A problem with using two very different software flows, such as the two depicted in **Fig. 1**, for computing a function is that a test and branch instruction, as in operation 102, is needed to implement the decision as to which flow to take. Even a single branch can cause severe performance penalties when a complex program is being executed. In certain modern computer architectures that have deep pipelines, and thus aggressive instruction prefetching, branch mispredictions can cause a large, filled pipeline to be drained, thereby rendering the instruction prefetching a waste. Also, if the architecture allows significant parallel data processing, branch-free table look-up implementations can offer a significant performance improvement.

BRIEF DESCRIPTION OF THE DRAWINGS

[0007] The invention is illustrated by way of example and not by way of limitation in the figures of the accompanying drawings in which like references indicate similar elements. It should be noted that references to "an" embodiment in this disclosure are not necessarily to the same embodiment, and they mean at least one.

[0008] **Fig. 1** shows a diagram of a conventional dual flow with branch methodology for computing $\log_b(X)$.

[0009] **Fig. 2** shows a diagram of a branch-free methodology for computing the logarithm function.

[0010] **Fig. 3** illustrates a diagram of a branch-free methodology for computing the natural logarithm.

[0011] **Fig. 4** depicts a diagram of a branch-free methodology for computing the base 10 logarithm function.

[0012] **Fig. 5** depicts a block diagram of a computer system for implementing the branch-free methodology of computing a transcendental function.

DETAILED DESCRIPTION

[0013] A computer-implemented method is described for approximating a function of an input argument. The method can be implemented by a single software flow, which helps eliminate heavy branch misprediction penalties associated with the conventional dual flow methodology. This branch-free methodology may be applied to compute transcendental functions such as the exponential, logarithm, and trigonometric functions. According to an embodiment of the invention, the conventional methodology is modified so that the lookup table has at least one breakpoint for which the reduced argument can be computed without roundoff error when the input argument is close to a root of the function. This modification helps avoid the loss of precision encountered during the right-hand flow of **Fig. 1** when the input argument is closer to 1 by less than, for instance, 2^{-9}. The modified breakpoint values allow the same flow to be used for all values of the input argument, even when close to a root of the function being evaluated.

[0014] Various embodiments of the branch-free methodology for computing a transcendental function are described below. These embodiments are based on representing X in the floating point form $Y*G^k$ where Y is greater than or equal to 1. It should be noted, however, that other floating point forms may be used such as one in which Y lies between 0 and 1. For conciseness, however, the following embodiments are described using only the format in which Y lies in the range 1 to 2.

Example: General Base Logarithm

[0015] **Fig. 2** shows a flow diagram, including operations 204-216, of a branch-free table-lookup methodology for computing $\log_b(X)$. The operations are discussed in more detail below.

[0016] Notation: Let the input argument be $X = 2^k \times Y$, $1 \leq Y < 2$. Let C approximate the value $\log_b e$ to a short precision. Let $B_0 > B_1 > \ldots > B_N$ be a set of breakpoints such that for all Y there exists a j such that $|YB_j - 1| \leq \text{delta} \approx 1/(2N)$.

[0017] Argument Reduction: Referring now to operation 204, find a suitable breakpoint B_j and compute the reduced argument $Z = C(YB_j - 1)$.

[0018] Core Approximation: Returning now to operation 212, let P(Z) estimate the approximate function $\log_b(1 + [Z/C]) - Z$ to a relative accuracy comparable to roughly 2^{-5} epsilon, where epsilon is the machine's unit roundoff, e.g. 2^{-24} for single precision or 2^{-53} for double precision.

[0019] Reconstruction: The final result is $k\log_b 2 + \log_b(1/B_j) + Z + p(Z)$, computed in an appropriate way so as to maintain a desired level of numerical precision. The values $\log_b 2$ and $\log_b(1/B_j)$ are computed beforehand and stored in a table (see operation 208).

[0020] Note that in the Core Approximation step described above, the approximate function is $\log_b(1 + [Z/C]) - Z$ in terms of the variable Z, instead of a more straightforward approximation function $\log_b(1 + R)$ in terms of the variable $R = YB_j - 1$ as used in conventional techniques to compute the logarithm.

[0021] In the methodology described above, it should be noted that the sequence of breakpoints $B_0, B_1 \ldots B_N$ is arranged from the smallest to the largest to more quickly yield the selection B_j that helps minimize the reduced argument $|YB_j - 1|$. Once the reduced argument Z has been computed, the core approximation may be performed by evaluating the approximate function $\log_b(1 + [Z/C]) - Z$ using, for instance, a conventional polynomial approximation in Z.

[0022] As mentioned above, one or both of B_0 and B_N may be set such that the reduced argument can be computed without roundoff error. Generally, either B_0 or B_N is selected as the breakpoint when X is close to a root of the function being evaluated. Thus, taking the logarithm function as an example, its root is at X = 1. Accordingly, as the value of X approaches 1, it may be expressed as either $1.000 \ldots 001 * 2^0$ or $1.999 \ldots 999 * 2^{-1}$. If X is expressed by the former, then the selected breakpoint is $B_0 = 1$ when $|Y-1|$ is less than delta. If the latter, then the breakpoint selected is $B_N = 1/2$ when $|Y-2|$ is less than delta.

[0023] To insure highest precision in approximating the function of X, the occurrence of roundoff errors should be minimized as much as possible. This may be accomplished by splitting each term of the approximate function into a pair of working-precision components whose sum is the value of that term. For the general base logarithm example given above, the value of $\log_b(2)$ is stored as a pair of working-precision numbers L_{hi} and L_{lo}, and the values of $\log_b(1/B_j)$ are stored in pairs of $T_{j,hi}$ and $T_{j,lo}$. For the example given here, precision is improved if $k L_{hi} + T_{j,hi}$ is representable exactly, that is without any roundoff errors, for all valid values of k and j. Consistent with $B_j = 0$, precision is also improved if $T_{0,hi} = T_{0,lo} = 0$. Also, consistent with $B_N = 1/2$, precision is further improved if $T_{N,hi} = L_{hi}$ and $TN_{,lo} = L_{lo}$. Finally, there can be a further improvement in precision if C, which approximates $\log_b e$, is represented so that $Z = C(YB_j - 1)$ is computed without roundoff error when j = 0 or j = N.

[0024] Together with the component representation described in the previous paragraph, precision is further improved if the computation sequence is as follows: First, the reduced argument C (YBj - 1) is computed as a pair of working-precision components Z_{hi} and Z_{lo} such that their sum approximates Z to higher than working-precision. In addition, they add up to Z exactly (without roundoff error) when j = 0 and j = *N*. Second, $kL_{hi} + T_{j,hi} + Z_{hi}$ should be representable exactly in working-precision for all valid values of k, j, and Z. This means that the components in the actual Reconstruction operation 216 include $A_1 = kL_{hi} + T_{j,hi} + Z_{hi}$, and $A_2 = kL_{lo} + T_{j,lo} + P(Z)$. The actual high accuracy computation is depicted in **Fig. 2** as a single program flow of operations 204-216 in which the index j has been dropped for clarity.

[0025] Referring to **Fig. 2**, it might at first appear that there is conditional code (and thus perhaps a test and branch instruction) in the Reconstruction operation 216, since there are two different additions, depending on whether $kL_{hi} + T_{j,hi} = kL_{lo} + T_{j,lo} = 0$, or otherwise. However in reality the different additions in operation 216 can be implemented in the same way except for different placements of the Z_{lo} term. This different placement can be implemented with conditional data movements which do not require test and branch instructions.

Example: Natural Logarithm

[0026] The following is a specific realization of the computation of a natural logarithm function in double precision. A flow diagram for this embodiment has operations 304, 308, 312, and 316 in **Fig. 3**. The operations 304-316 may be generally similar to 204-216 of **Fig. 2**, except that the logarithm is for base e.

[0027] Breakpoint Definition and Argument Reduction: Let the input argument be

$$X = 2^k \times Y,\ Y = 1.y_1y_2y_3y_4y_5y_6\ldots y_L = 1 + i/32 + \text{beta},\ 0 \le \text{beta} < 1/32.$$

Define the auxiliary values $F = 1 + i/32$ if $\text{beta} < 1/64$, and $F = 1 + (i + 1)/32$ if $\text{beta} \ge 1/64$. Hence, $F = 1 + j/32 = F_j$, $j = 0, 1, \ldots, 32$. Define the breakpoints $B_j = 1/F_j$ rounded to finite precision with 10 significant bits. Recall that $B_0 = 1$ and $B_{32} = 1/2$. Note that the index j is really given by $[y_1y_2y_3y_4y_5] + y_6$. By masking of binary bits, decompose Y into two working-precision variables Y_{hi} and Y_{lo} where Y_{hi} is Y with the lower 32 significant bits set to zero. C is 1 in this case. The several components of the reduced arguments are computed as $Z_{hi} \leftarrow Y_{hi} B_j - 1$, $Z_{lo} \leftarrow Y_{lo} B_j$, $Z \leftarrow Z_{hi} + Z_{lo}$.

[0028] Table Value Calculations: The leading parts of T_{hi} and L_{hi} of the table values are all obtained by rounding the ideal values to a precision such that the least significant bit is 2^{-43}. Hence, $L_{hi} = \log_e 2$ is rounded to lsb at 2^{-42}, and $T_{j,hi} = \log_e (1/B_j)$ is similarily rounded. The trailing parts are simply the working-precision approximation of the differences between the ideal values

and the leading values. Hence $L_{lo} = \log_e 2 - L_{hi}$ is rounded to 53 significant bits, and $T_{j,lo} = \log_e (1/B_j) - T_{j,hi}$ is rounded to 53 significant bits.

Example: Base 10 Logarithm

[0029] An embodiment of the branch-free methodology for the base-10 logarithm function is as follows. See also the flow diagram in **Fig. 4** in which operations 404, 408, 412, and 416 are depicted.

[0030] Breakpoint Definition and Argument Reduction: Let the input argument be

$$X = 2^k \times Y, \; Y = 1.y_1y_2y_3y_4y_5y_6\ldots y_L = 1 + i/32 + \text{beta}, \; 0 \le \text{beta} < 1/32.$$

Define the auxiliary values $F = 1 + i/32$ if $\text{beta} < 1/64$, and $F = 1 + (i + 1)/32$ if $\text{beta} \le 1/64$. Hence $F = 1 + j/32 = F_j$, $j = 0, 1, \ldots, 32$. Define the breakpoints $B_j = 1/F_j$ rounded to finite precision with 10 significant bits. Recall that $B_0 = 1$ and $B_{32} = 1/2$. Note that the index j is really given by $[y_1y_2y_3y_4y_5] + y_6$. By masking of binary bits, decompose Y into two working-precision variables Y_{hi} and Y_{lo} where Y_{hi} is Y with the lower 32 significant bits set to zero. Pick C to be 28/64 in this case, which is a 5-significant-bit approximation of $\log_{10}(e)$. Instead of storing B_j, store the values $D_j = CB_j$, $j = 0, 1, \ldots, 32$. The several components of the reduced arguments are computed as $Z_{hi} \leftarrow Y_{hi} D_j - C$, $Z_{lo} \leftarrow Y_{lo} D_j$, $Z \leftarrow Z_{hi} + Z_{lo}$.

[0031] Table Value Calculations: The leading parts of the table values are all obtained by rounding the ideal values to a precision such that the least significant bit is 2^{-43}. Hence, $L_{hi} = \log_e 2$ rounded to lsb at 2^{-43}, and $T_{j,Hi} = \log_b (1/B_j)$ is similarly rounded. The trailing parts are simply the working-precision approximation of the differences between the ideal values and the leading values. Hence $L_{lo} = \log_e 2 - L_{hi}$ rounded to 53 significant bits, and $T_{j,lo} = \log_e (1/B_j) - T_{j,hi}$ rounded to 53 significant bits.

[0032] The various embodiments of the branch-free methodology described above avoid the conventional numerical problems of table-lookup techniques which occur near the root of the transcendental function. Since there are a relatively small number of table values that create this numerical imprecision near the root of the transcendental function, where in the above examples it was one or

both of the endpoint values B_0 and B_N, the branch-free methodology ensures that these small number of values are exact, such that no roundoff errors are present. For instance, in the case of the logarithm function, this was insured by setting $B_0 = 1$ such that the table value is 0. Alternatively, this could be insured by setting $B_N = 1/2$ and the table value there is exactly that stored for $\log_b(2)$ such that the terms in the approximate function that include B_N and $\log_b 2$ cancel each other when $k = -1$, leading to exactness. The approximate function estimates the desired transcendental function using a reduced argument, in a small region around the root of the transcendental function. For instance, in the case of the logarithm function, the reduced argument is $Z = C(YB_j - 1)$. Moreover, for highest accuracy, this reduced argument should be computed to an accuracy higher than that of working precision. Thus, in the case of the logarithm function, this can be obtained by computing the reduced argument as components Z_{hi} and Z_{lo}. In addition, to obtain the highest accuracy, the components should be combined with the table values in a way that preserves the extra accuracy that was computed for the reduced argument. Again, in the case of the logarithm function, this may be insured by arranging the value $kL_{hi} + T_{hi} + Z_{hi}$ to be exactly representable.

[0033] The concepts of the various embodiments of the branch-free methodologies described above are also applicable to other transcendental functions. One example is the computation of the function exp (X) - 1 over a small range around its root, 0. In that case, an exemplary set of table values would correspond to $\exp(j/2^m)$ for some m. The reduced argument of the approximate function in this case would have the form $X - (j \log 2)/2^m$.

[0034] Another example for computing a transcendental function using the branch-free software methodology is the computation of atan (X). Here, the table values can be atan (B) for some breakpoint B that approximates X. The reduced argument of the approximate function can be of the form $Z = (X-B)/(1+BX)$.

[0035] **Fig. 5** shows a block diagram of a computer system 502 that may be configured with instructions that when executed by a processor approximate a transcendental function according to the branch-free methodology. The system 502 features a processor 504 that is coupled to a nonvolatile mass storage device 514 via a bus 526. The mass storage device 514 may be a conventional rotating

magnetic disk drive or other nonvolatile memory for storing program instructions and data to be executed by the processor 504. Instructions and data are normally transferred to program memory 508, which may be a higher speed, volatile memory such as dynamic random access memory (DRAM), as they are executed by the processor 504. The results of the execution may be displayed using a display 522, such as a cathode ray tube (CRT) or other visual display device, accessed via a display interface 518. In addition, the results of the program execution may be transferred out to a data network via a network interface 512. The program instructions and data for the branch-free software methodology are introduced into the system 502 via either the network interface 512 or through a portable storage device interface 510. The latter acts as an interface to a storage medium such as a compact disc read only memory (CD-ROM) or other portable, nonvolatile storage device.

[0036] As mentioned above, the branch-free software methodology for computing a transcendental function would normally be written in assembly language code that is specific to the processor 504 of the system 502 (see **Fig. 5**). The assembly language code may be sold as part of a compiler program to translate higher level programs, written in a particular source code, into machine code for the particular processor 504. The compiler program would include an operation in which the source code is parsed according to conventional techniques and a reference to a transcendental function is detected. The compiler may then replace all instances of this high level function call by a sequence of assembly language instructions that implement the appropriate branch-free methodology for that transcendental function.

[0037] To summarize, various embodiments of a branch-free methodology for computing a transcendental function have been described. In the foregoing specification, the invention has been described with reference to specific exemplary embodiments thereof. It will, however, be evident that various modifications and changes may be made thereto without departing from the broader spirit and scope of the invention as set forth in the appended claims. The specification and drawings are, accordingly, to be regarded in an illustrative rather than a restrictive sense.